

# A FLEXIBLE APPROACH TO AUTHORIZATION OF UAS SOFTWARE

*P. Graydon, J. Knight, K. Wasson*

*Department of Computer Science, University of Virginia, Charlottesville, VA*

## Abstract

Unmanned Aircraft Systems (UASs) rely upon a significant amount of software. An appropriate authorizing agent must approve the use of the UAS in a desired environment before use, and the authorization approach used must contend with the UAS' software. Unfortunately, the variety of UAS types, the range of environments within which they must operate, and the need to address both safety and security concerns make authorization of UAS software problematic.

We have developed a flexible approach to UAS software authorization that is able to deal with these challenges. Our approach is based on rigorous *fitness arguments* that explain how evidence from the software's development shows that the software has the properties that make the UAS fit for use in the intended operating contexts.

In this paper, we present the details of our approach, compare it to existing approaches, and show how retroactive construction of software fitness arguments can identify the additional evidence necessary to support full authorization or the limited authorization that can be granted based on existing evidence. We give examples to illustrate how our approach can be used across a wide variety of UASs, missions, and operating environments, including controlled airspace.

## Introduction

Unmanned Aircraft Systems (UASs) rely upon a significant amount of software for operational functionality, payload management, and safety. Before a UAS is deployed, the use of the UAS in the desired environment must be authorized, typically by an agent independent of the UAS' developers. That is, an appropriate authorizing agent, after conducting any needed analysis and assessment, must approve operation of the aircraft in the designated environment before that operation commences. Since UASs rely upon software, any authorization mechanism must address the aircraft's software.

The variety of UAS types and the range of environments within which they must operate make UAS software authorization problematic. Different techniques are needed for different aircraft and even for the same aircraft in different operating environments.

To deal with this issue, we have developed a flexible approach to UAS software authorization that is able to deal with all the challenges, including the need for both safety and security. Rather than require that the software be developed in a particular way, our approach focuses on a rigorous software *fitness argument*, ideally developed by the aircraft's manufacturer and supplied to the authorizing agency.

A software fitness argument is a compelling rigorous argument that the software possesses the characteristics that it must have if the UAS in which it is used is to be acceptably safe and secure to operate in specified environments. Similar in notation, form and concept to the safety arguments used in other domains [1], the software fitness argument collects and explains the relevant and available evidence from both the UAS software artifacts and the processes by which they were developed, showing how this evidence contributes to the conclusion that the software is fit for use.

In this paper, we present the details of our approach to UAS software authorization, and we compare the approach to existing regulatory techniques. We describe how *retroactive* construction of software fitness arguments permits authorizing agents to cope with software for which insufficient evidence is available from the manufacturer. In particular, we present details of the techniques used to: (1) permit authorization in limited operational environments; and (2) determine the complete set of evidence required to support operation in the desired environment. We further give examples to illustrate how our approach can be used across a wide variety of UASs, missions, and operating environments, including the authorization

of software for UAS operations in controlled airspace.

## The UAS Software Challenge

Unmanned air systems are rapidly growing in number, application and design complexity. They operate in both military and civil domains, and software authorization for these systems poses a host of new challenges to organizations responsible for this task. These challenges arise in part from:

- The diversity of system and software architectures that are used in the aircraft. Some aircraft use extensive distributed computer systems, advanced operating systems, and sophisticated applications. Others use a minimal computer system with only elementary operating systems and simple applications.
- The wide range of environments these systems operate in. Aircraft operate, or will operate, over uninhabited regions, oceans, heavily populated areas, and theaters of conflict.
- A broad array of aircraft configurations ranging from electrically-driven aircraft to turbine-engined aircraft, from masses of a few kilograms to masses of thousands of kilograms, from speeds of a few tens of knots to several hundreds of knots, and from carrying camera platforms to carrying weapons.
- The diversity of missions for which UASs can be deployed, and the corresponding diversity in the cost of a mission failure.
- The context and constraints imposed by existing organizational infrastructures of manufacturers, owners and authorizing agencies.

Broadening procedural pressures add further challenges, including:

- The need to authorize software for a wide variety of systems through a single comprehensive process.
- The need to authorize systems already in operation.

- The need to authorize software in the absence of some or all information about how the software was developed.
- The need for rapid authorization in order to accommodate the demands of conflict support.
- The need to avoid unduly burdening either developers or authorizing agents.

Software support for system safety and other desirable properties is more difficult to demonstrate under these increasingly complex constraints, and existing certification processes are being stretched beyond their design envelope in handling the volume and complexity of systems requesting software authorization.

UAS software authorization presents challenges that do not arise with manned aircraft. These challenges stem from the wide variety of types, technologies, sizes, weights, and payloads of current and planned UAS, and the wide variety of desired missions and operational environments. This variety drives a corresponding variety in amount, form, complexity, and criticality of aircraft software.

Authorizing agents have already developed and used techniques for authorizing software for use on manned aircraft. These approaches, however, are based on a narrow risk model that is inapplicable to the full variety of UAS and missions with which authorizing agents must cope. While the primary safety concern with manned aircraft is avoiding damage to the aircraft and possible harm to humans, the variety of UAS and mission types means that, in some circumstances, damage to or loss of the aircraft might be preferred over other outcomes. Crashing the aircraft into uninhabited terrain, for example, might be preferable to risking a crash into a heavily populated area or to allowing takeover of the aircraft by a malicious adversary.

Software authorization approaches for manned aircraft also frequently require the carefully monitored use of prescribed development practices such as those demanded by RTCA DO-178B for commercial aircraft software [2]. While such expensive approaches might be appropriate for some UASs and operating environments, they are clearly inappropriate for others.

A requirement that is immediate and challenging for UASs but which is only emerging for manned aircraft is cyber security. Dealing with security as well as all the other requirements areas for UASs is technically difficult because of the conflicts that sometimes arise in the design of software between support for system safety or functional goals and support for security goals. Assessment for purposes of authorization is, therefore, especially problematic.

We assert that the introduction of *fitness arguments* into the process of UAS software authorization can function as an effective solution to the current authorization challenges. The notion of an assurance argument (of which both safety arguments and fitness arguments are special cases) is not new. Safety arguments, in particular, have been used successfully as an enabling technology for building stakeholder confidence that a system is adequately safe to operate. We propose that fitness arguments can also be used by organizations other than OEMs and for wider purposes.

In our approach, software authorizing agencies use fitness arguments to better structure the activities involved in authorizing a software system for operation. Also, such agencies can use the fitness argument to make clear and provide rational support for their decisions about partial clearances for software systems where previously such support has been difficult or not possible. Further, authorizing agents will be able to communicate back to OEMs exactly which kinds of remedies are necessary in order to gain full desired clearances.

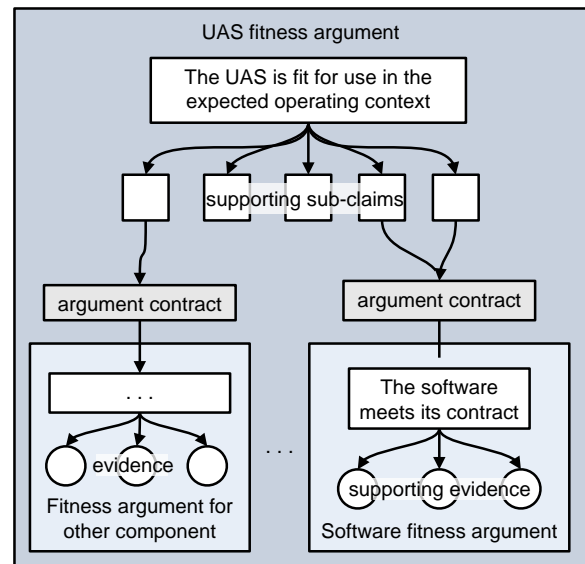
## UAS Systems vs. UAS Software

The software *component* of a UAS is an abstract mathematical entity. As such, it is absurd to speak of the safety or security of the software in isolation; we can only sensibly discuss these as attributes of the complete unmanned aerial *system*. Thus, the goal in software authorization is to determine whether a UAS' software has the characteristics that the UAS depends upon for safe and secure operation.

In our approach, engineers with appropriate *systems-engineering* expertise create a documented fitness argument for the complete UAS. Each UAS fitness argument identifies the UAS and the contexts in which it will be operated and presents

both a justified enumeration of the safety hazards and security threats to be avoided and a structured, rational argument showing that each hazard and threat has been adequately mitigated. Ideally, the system developers would create this argument. However, as we will show, our authorizations approach may be used even when the manufacturer has not supplied an argument.

This system-level argument will contain one or more premises that are themselves claims about the UAS' software. These claims form an *argument contract* that specifies what the system will count upon the computing system to do and thereby what the software developers must ensure that the computing system does. Suppose, for example, that for a given UAS to be fit for use given the system-level design and the properties of its other components, its autopilot software, running on the target CPU, must achieve a prescribed reliability. Achievement of a commensurate degree of certainty that the software is free of defects would be one of the claims in the autopilot software's argument contract. This arrangement is illustrated in **Figure 1**.



**Figure 1. Software, systems, and contracts**

## Software Fitness Arguments

A fitness argument is a generalization of the concept of a safety argument to support a broader goal. A safety argument is:

*“...a clear, comprehensive and defensible argument that a system is acceptably safe to operate in a particular context” [1]*

A fitness argument has the form of a safety argument, but with a different goal. Rather than support a claim that a system is *acceptably safe to operate* in given contexts, it gives a rationale for believing that the system is *fit for use* in the given contexts. We deliberately define fitness for use broadly: in our view, a system is fit for use if it demonstrably meets an acceptable balance of stakeholder concerns including functionality, safety, and security.

A fitness argument for a software component explains a rationale for believing that the software is fit for use in the wider context of the UAS(s) within which it will be used. Since what is meant by fitness for use in a given UAS is defined by the argument contract(s) between the software and the UAS(s), a software fitness argument demonstrates how the available evidence supports the conclusion that the software satisfies all applicable contracts that the system expects to be met.

The evidence that appears in a software fitness argument consists of details of:

- how the software was constructed;
- the software’s design;
- the test plan and results;
- the results of static analyses; and
- any other relevant information about the software’s construction or behavior.

Much, if not all, of this information is described in other system artifacts: test plans and documentation, design documentation, etc. The crucial role of the argument is to collect this information and explain how it *collectively* shows that the software is fit for use. As Kelly said of safety arguments:

*“Argument without supporting evidence is unfounded, and therefore unconvincing. Evidence without argument is unexplained — it can be unclear that (or how) safety objectives have been satisfied” [3].*

Safety and other assurance arguments have been recorded in a variety of notations, including Adelaar’s graphical Claims Argument Evidence

(ASCAD) notation [4] and natural language text. In our own work with fitness arguments, we have used Kelly’s *Goal Structuring Notation* (GSN) [1] diagrams.

The notion of argument is well established and its use in safety discussed broadly in the literature, and so we do not discuss the details of argument further. Once familiarity is established by developers and authorizing agents, the use of argument in a rigorous manner will facilitate more rapid and effective communication between all parties involved in software authorization.

## UAS Software Authorization

### *Artifacts of the Approach*

The architecture of the software authorization approach that we are developing is shown in **Figure 1**. At the highest level of abstraction, there are two major artifacts: a fitness argument for the *overall system* and a fitness argument for the *software*. The link between the two is the set of software requirements.

There are two major components of the software requirements: (1) the functional requirements; and (2) the dependability requirements. The functional requirements are derived from the system’s design. This notion is quite familiar, and so we will not discuss it further.

The dependability requirements define attributes of the software such as reliability, availability or security, and they derive from the system fitness argument. In other words, in order to be valid, the *system’s* fitness argument depends upon the *software* meeting its dependability requirements. The reason we label the software requirements as the argument contract is because meeting those requirements is a necessity for the system’s argument.

As an example, consider the software for a UAS flight-control system, a complex hard-real-time system. If there is no mechanical backup, then controlled flight depends on the computerized flight-control system working correctly. In that case, the safety argument for the complete UAS system will require evidence that the flight-control system’s software is ultra-dependable. However, if the system architecture includes a mechanical backup, then the software need only be moderately

reliable because a software failure would not be catastrophic. Thus the contract that the software has to meet depends upon what is needed in the safety argument for the system which itself depends on the system design.

### ***Mechanism of Authorization***

The core of our approach to UAS software authorization is for the authorizing agent to determine whether the software complies with the argument contract based on the associated fitness argument. To do so, the agent examines the software's fitness argument and determines whether the argument makes a complete and compelling case that the software complies with the contract. If it does, then the software is authorized. The process of examining the software's fitness argument is one of inspection and can be made a routine and systematic process that is amenable to tool support.

The argument contract provides the critical link between system assurance and software assurance. The form of this mechanism is important for two reasons:

- UAS domain experts will be responsible for the UAS design and the associated UAS system fitness argument. Similarly, software experts will be responsible for the software development and the software fitness argument. The contract mechanism ensures that software experts do not have to be responsible for analyzing system safety issues, an area in which they are unlikely to be specialists.
- The assurance contract is specific to the particular UAS of interest and its intended operating contexts. Thus, the precise safety and security circumstances of each UAS can influence its argument contract and thereby the dependability requirements of its software. The result is a high level of flexibility that permits a wide range of aircraft and mission scenarios as discussed earlier to be accommodated.

Some portions of the argument contract may support only system fitness argument claims other than safety or security, and some authorizing agents may be unconcerned with UAS properties such as usefulness. Since many safety- and security-derived

requirements will call for the software to perform its intended functions dependably, however, we expect the majority of each argument contract to interest most agents. Accordingly, we do not ask argument writers to excerpt "relevant" portions for approval.

### ***Retroactive Argument Update***

When authorization for a UAS and its software is sought, the authorizing agent might discover one of two obstacles to authorization:

- The developer of the UAS' software has not prepared an explicit fitness argument and so there is no explicit basis for authorization.
- The developer of the UAS' software has prepared a fitness argument but the authorizing agent does not find the argument compelling.

The possibility of either of these outcomes can be reduced by: (a) the publication of appropriate guidance for constructing arguments and navigating the authorization process; and (b) by requiring developers to draft arguments early and update them frequently. Nevertheless, practical realities stemming from both the diverse nature of UAS manufacturers and from the logistical impact of adopting an authorization mechanism dictate that such situations will occur.

In the first obstacle above, the UAS developer had an informal, ad hoc and undocumented rationale in mind for believing that the software's dependability would meet the system's needs. Otherwise the developer would have no reason to believe that the aircraft would operate correctly at all. Most likely, this informal fitness argument was based on the assumption that the presence of common kinds of evidence, such as test results, is a sufficient basis upon which to conclude that the system is fit for use. Unless the argument is made explicitly, however, we cannot be sure that it is complete and logically valid.

In the second obstacle above, the argument is available but not compelling, and so the issue becomes one of belief in the argument. In essence, the first case is just a degenerate instance of the second case, and so we do not consider it separately.

To obtain proper authorization, the fitness argument for the software has to be made adequate. There are three possible approaches to achieving this:

- **Improve the Argument.** If the software was in fact adequate but the argument was in some way deficient, the software fitness argument could be improved to make the argument complete and compelling. The problem might have arisen, for example, because there was insufficient evidence to construct a compelling argument or because of defects in the argument.
- **Improve the Software.** Improving the software involves changing the software so as to allow stronger evidence to be obtained and used in the software fitness argument. This stronger evidence might be, for example, the results of additional testing, new or enhanced documentation, or design changes together with associated analysis.
- **Weaken the Contract.** The software and its fitness argument could remain unchanged and the contract weakened until the argument is sufficient. If the contract is to be weakened, then the system fitness argument has to be changed so that the weakened contract *is* adequate to support the system fitness argument. The system fitness argument could be modified, for example, to permit operation in a less challenging operating environment such as operation over unpopulated areas only, to restrict operational conditions such as operation only within visual range, or by accepting a higher level of risk such as a higher probability of aircraft loss.

These three approaches are the basis for the operational use of the authorization technique that we are developing. They present the UAS developer with clear choices that can be pursued so as to allow tailored UAS software authorization for any aircraft under any desired circumstances.

### ***Summary of Approach***

In summary, our approach consists of the use of a rigorous fitness argument to document the rationale that the developers of a UAS software system have for believing that the software meets the requirements imposed by the system fitness argument. We anticipate this approach being effected in practice in one of four different ways:

- Developers deliver the UAS software and deliver a complete argument in an approved notation with the software. Provided that the argument is compelling, authorization of the software is likely.
- Developers deliver the UAS software and deliver an incomplete argument with the software. The software will not be authorized, but the partial argument will allow diagnosis of the steps necessary to achieve authorization.
- Developers deliver the UAS software with no argument. The software will not be authorized, and the developers and authorizing agent will work together to develop an appropriate fitness argument in an approved notation. This and the previous circumstance are likely to occur as developers get used to the approach.
- Developers create a *reusable* software *component* for use in UASs (not a specific system for a particular aircraft) together with a reusable fitness argument fragment for the software component. Those building UASs can then employ the component and create a fitness argument for the complete software system incorporating the supplied fitness argument fragment for the component.

This approach accommodates any combination of UAS, mission, operational requirements, and development methodology provided that the developers can argue convincingly that the result is acceptably fit for use. In addition, the flexibility of the technique permits the introduction, routinely, of new technology as it is developed. There is no need for developers to wait for the update to a prescribed standard.

## Existing Authorization Approaches

Several organizations are charged with authorizing UAS software for different aircraft, different operators, and different missions. The approaches that they use share certain features, and we discuss each of these features in turn.

### *Prescribed Processes*

Many of the existing approaches to authorizing software for manned aircraft call for developers to follow a prescribed process. When the principal hazard to be avoided is loss of the hull, a prescriptive standard can be parameterized by how likely a failure of the computing system is to cause such a loss. DO-178B, for example, is organized in this manner.

The wide variety of UAS types and missions, however, precludes such a simple hazard model. The risk associated with air-to-air collision varies with the airspace in which the UAS is operated. Likewise, the risk associated with the air-to-ground collision varies with the population density on the ground and with aircraft size, mass, speed, energy source, and payload.

In addition, the fact that an aircraft in an UAS is designed to allow remote operation presents a security threat. Without a pilot on board the aircraft to detect undesired operation and take manual control, a UAS' software must be relied upon to address security threats adequately. The nature and consequence of the threats is as diverse as the missions and operating contexts of the UASs they bear upon.

When these diverse concerns are decomposed across the system architecture, the result is a wide variety of software dependability requirements. A system might require high confidence in some functions but not in others, or uniformly high or low assurance across functions. Some systems might require substantial evidence of non-interference between the implementations of each function. Some might require strong evidence of freedom from certain security vulnerabilities, while others might be more concerned with overall availability.

Parameterizing a prescribed process to account for all expected variations, while not impossible, is nevertheless difficult. An approach based upon software fitness arguments does not suffer from this

problem because it focuses on demonstrating that the software has precisely the qualities needed for the UAS to be fit for use.

Perhaps more importantly, authorization based on fitness arguments permits the use of novel software engineering techniques as they become available. While a developer subject to a prescriptive standard who wishes to use a model-based development tool, for example, must wait for the standard to be updated to discuss such tools, a developer seeking argument-based authorization is free to use them right away provided that a compelling argument can be made that the new technique contributes properly to the software's fitness for use.

### *Checklists*

Other existing approaches feature checklists. In essence, this is a degenerate case of a prescriptive process. The authorizing agent decides whether each item is relevant to a given UAS, and, if so, whether the concern it raises or process element it mandates has been addressed or performed.

This approach allows the authorization process to be customized to the unique circumstances of each UAS. However, even if the authorizing agent is required to justify each decision that an item is not relevant, the overall rationale for claiming that the software is fit for use in a given UAS in given context(s) is left implicit. We contend that the reason for belief that compliance with the items not deemed irrelevant implies fitness for use should be made explicit, and that fitness arguments are an effective way to accomplish that.

### *Software Authorization Levels*

Some existing approaches to UAS software authorization are based on *authorization levels* [2]. Typically three or four levels are used where each level addresses more rigorous demands than the one below it. This approach is an example of the broader technique of *software integrity levels* (SILs) [5].

While the use of software integrity levels helpfully compartmentalizes the problem, this approach is still insufficiently flexible for the wide spectrum of UAS and operating environments. Any

specific UAS has to be associated with one of a small number of levels, and the result will usually be software that meets some of the UAS' needs and not others.

For example, consider a small aircraft in which weight, size, and power constraints dictate that several software components will share a single computer. Suppose that some software functions, such as control of the flight surfaces, are critical while others, such as camera control and image processing, are not. A SIL-based approach might dictate that the entire software package be developed with a degree of rigor appropriate for a high SIL level. An argument-based approach, on the other hand, permits the developer to use less expensive procedures on the code implementing the lower-criticality functions so long as adequate evidence of non-interference can be supplied.

## Illustrative Examples

In this section, we present a series of *hypothetical* examples to illustrate how the approach we are developing would address the challenges that UAS software authorization faces. The first five examples are a UAS system for monitoring over a heavily populated area. The examples differ in the computer functionality, the computer system architecture, and the software technology used.

### *Monitoring Populated Area - 1*

#### Scenario

Aircraft application:	Monitoring critical government installation
Operating airspace:	Populated areas, not controlled airspace
Flight duration:	24 hours
Flight profile:	Autonomous navigation, pilot monitoring
Payload:	Cameras & other sensors
Weapons:	None
Mass:	5,000 kg
Speed:	400 knots
Computer system:	Distributed, fault tolerant computers, AFDX data bus, custom software

## Argument Contract

From the system design, the systems engineers defined the following aircraft operational services that the software must provide:

- Flight control
- Flight management
- Altitude hold autopilot
- Return to rendezvous location
- Return to crashport and self destruct

These functions are safety critical, and so the system fitness argument requires extremely high levels of software reliability (e.g.,  $10^{-9}$  failures per hour of operation). In addition, the aircraft's application and potential for causing damage dictate that the command link is security critical.

## Engineering Process

The software engineering team created the software, the evidence and the fitness argument that the software meets its contract. The software technology used includes:

- Formal specification of critical functions.
- Formal verification of crucial properties.
- Testing to MC/DC coverage.
- Comprehensive traceability analysis.
- Thorough static analysis of the source programs.
- Complete analysis of the worst case execution times of the various software components together with proofs of real-time schedulability.
- Detailed software security audit.
- Comprehensive use of security technology on the command link including authentication and encryption with asymmetric keys.

## Software Authorization

The argument supplied to the authorizing agent was deemed compelling. The strategy used within the system fitness argument was that all credible safety hazards and security threats had been enumerated and been mitigated to an appropriate level. The software fitness argument showed that the software had the necessary properties to support this line of argument. The aircraft software was authorized for operational use.

## Monitoring Populated Area - 2

### Scenario

This scenario is identical to the previous one but a sense-and-avoid capability based on a commercial off-the-shelf software component is added. The sense-and-avoid software runs on the same computer hardware platform as the avionics applications.

### Argument Contract

The aircraft's software functionality derived from the system design is identical but with the addition of sense-and-avoid functionality.

The sense-and-avoid software only has to demonstrate moderate reliability, because there is a monitoring pilot. However, the new functionality is provided by a COTS component, and so the system fitness argument demands that the software demonstrate *non-interference* with high reliability. The concern expressed by the system engineers is that the COTS component could interfere with basic aircraft control.

### Engineering Process

The new component was integrated with the other parts of the system, but there was no comprehensive documentation available about the COTS sense-and-avoid software. That software was tested using functional testing only.

Non-interference is not guaranteed by the custom avionics software, and so the developers had to resort to testing in an effort to demonstrate the necessary non-interference requirement.

### Software Authorization

The argument supplied to the authorizing agent was *not* deemed compelling because of insufficient evidence of non-interference. Testing was not regarded as sufficiently strong evidence of the claim of non-interference in the software fitness argument. The software was not authorized for use.

In order to illustrate how the denial decision is made, Figure 2 shows a fragment of the supplied fitness argument. The fragment shown is part of the sub-argument for the goal "Sense-and-avoid software does not interfere with primary avionics system." In the complete argument, the strategy used by the developers was to argue over all possible mechanisms by which interference might occur that such interference had been shown to be

possible with a sufficiently small probability. There are five ways in which interference might happen:

- The sense-and-avoid software might fail to relinquish the processor (enter an infinite loop, for example) and thereby cause the primary avionics system to miss real-time deadlines.
- The sense-and-avoid software might write to memory being used by the primary avionics and thereby damage code or data of the primary system.

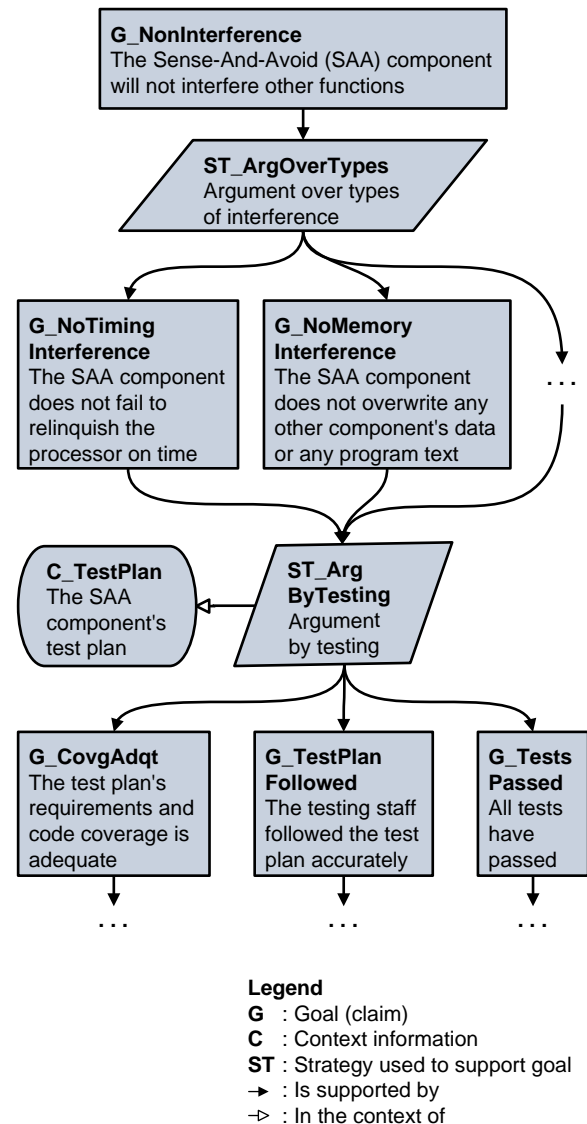


Figure 2 Testing-based independence argument

- The sense-and-avoid software might branch to a location outside of its own

instructions and thereby initiate some form of unplanned action.

- The sense-and-avoid software might write to a peripheral (an actuator, for example) that it was not supposed to access.
- The sense and avoid software might write to a hardware configuration register and thereby change how the hardware operates.

In Figure 2, we only show a fragment of the argument (for the first two interference mechanisms) starting with the claim of non-interference and elaborating the argument down two levels. Realistic arguments are much larger than can be presented here. Sub-arguments need to be developed for each one of the five types of interference so as to properly claim that each source is properly dealt with.

The software authorization failed because the authorizing agent did not find that sub-arguments based on testing were compelling. In practice, this is the right decision because the argument for each issue needs to be very strong. Adequately testing to show a suitably low probability of occurrence of these issues is impossible [6].

### Monitoring Populated Area - 3

#### Scenario

This scenario is identical to the previous one. The aircraft developers treat the reasons for their argument's rejection as a diagnosis of the additional work that is needed on the software. They undertake to develop the necessary evidence for the software fitness argument. This is referred to as *argument repair*.

#### Argument Contract

The aircraft operational services derived from the system design are identical, and the system fitness argument demands the same levels of reliability for the avionics and for the lack of interference.

#### Engineering Process

The software engineering process for argument repair required changes to the software, generation of new evidence, and revision to the fitness argument.

The developers decided to operate the sense-and-avoid system as an isolated software process in its own protected memory, and to prove that this architecture provides sufficient evidence that the sense-and-avoid system will not interfere with the remainder of the avionics.

#### Software Authorization

The *revised* software fitness argument, including details of the new software architecture and the new proofs, was supplied to the authorizing agent and was deemed compelling. The aircraft software and hardware were authorized for operational use.

The fragment of the revised argument based on the engineering changes (architectural change to an isolated process and proof of freedom from interference within the primary avionics) is shown in Figure 3. The sub-goals on testing have been replaced using the new evidence.

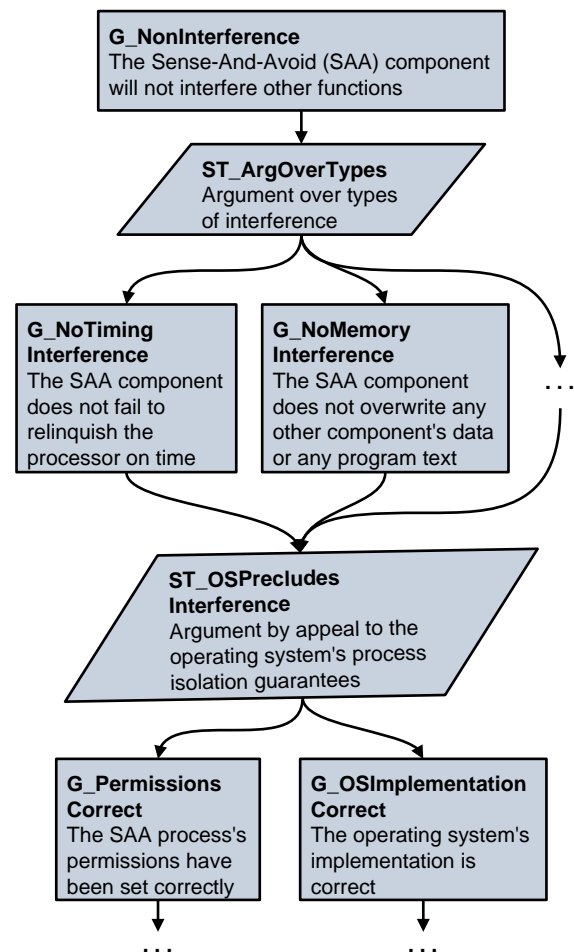


Figure 3 Arguing OS-enforced isolation

## ***Monitoring Populated Area - 4***

### **Scenario**

This scenario is identical to the previous one but with a different computing platform:

Computer system: Single computer only using a COTS operating system

### **Argument Contract**

The aircraft operational services derived from the system design are identical, and the system fitness argument demands the same high level of reliability.

### **Engineering Process**

The software engineering process was identical.

### **Software Authorization**

The software fitness argument supplied to the authorizing agent was deemed compelling but the associated argument about the computing hardware platform was not. Execution on a single computer was not deemed sufficient to guarantee the computer system levels of reliability needed. The necessary evidence that the computing platform would have ultra high availability could not be provided.

The solution in this case was to weaken the argument contract. The top-level goal of the system fitness argument was weakened by restricting the aircraft to operation over unpopulated areas and with a pilot in control at all times. This weakening of the top-level goal reduced the demands that the system fitness argument imposed on the computer system to one where uncontrolled loss of the aircraft would do little damage. With that restriction, the aircraft software and hardware were authorized for operational use.

## ***Monitoring Populated Area - 5***

### **Scenario**

The scenario is identical to the first one, but an existing UAS is selected for use. Software authorization is sought for the existing aircraft.

### **Argument Contract**

No contract existed because no fitness arguments were prepared for either the system or the software.

### **Engineering Process**

An ad hoc engineering process was followed for all aspects of development of the subject UAS. The aircraft has been in use in several operational contexts, and 5,000 hours of operational use had been accumulated with no loss of aircraft. The developers argued that this operational experience was sufficient to permit use in the population monitoring application.

### **Software Authorization**

Since no software fitness argument was supplied, authorization of the software was denied. The developers claimed that the operational use was compelling evidence of safe operation, but this argument was rejected.

In order to permit limited operation without incurring substantial delay and expense to reengineer the software, the available evidence was documented and both system and software fitness arguments were developed *retroactively*. Both were minimal, and both had only operational experience as evidence. Both treated this experience as testing, and a software fitness argument was created based on a statistical model of reliability assessment.

Because testing evidence was insufficient to support the desired operation, the top-level system fitness goal was weakened to one that could be met with the available evidence. With these restrictions in place, the software was authorized for use in the system during operation over unpopulated areas with a control pilot constantly in charge of the aircraft.

## ***Operation in Controlled Airspace***

### **Scenario**

Aircraft application: Monitoring light aircraft movements near an airport  
Operating airspace: Controlled airspace  
Flight duration: 24 hours  
Flight profile: Autonomous navigation, pilot monitoring  
Payload: Cameras & radars  
Weapons: None  
Mass: 1,000 kg  
Speed: 100 knots  
Computer system: Single fault tolerant computer, custom software

## Argument Contract

From the system design, the systems engineers defined the following aircraft operational services that the software must provide:

- Flight control
- Flight management
- Autopilot altitude hold
- Autopilot heading hold
- Return to rendezvous location
- Sense-and-avoid using on-board radars

These functions are safety critical, and so, once again, the system fitness argument requires extremely high levels of software reliability. In addition, the aircraft's application is an attractive target for terrorists, and so the command link is security critical.

## Engineering Process

The software development team was aware that the approach to software authorization would be based on their supplying a comprehensive fitness argument. That necessity was used to drive their software development approach.

The software engineering team created in parallel the software, the evidence, and a fitness argument showing that the software meets its contract. The software technology used includes:

- Compliance with RTCA DO-178B.
- Systematic use of formal specification.
- Systematic use of formal verification.
- Detailed software security audit.
- Comprehensive use of security technology on the command link including authentication and encryption with asymmetric keys.

## Software Authorization

The software fitness argument supplied to the authorizing agent included two sub-arguments beneath the top-level fitness claim. The argument strategy used was that either of the sub-arguments was sufficient to warrant operation in controlled airspace, and that their composition allowed for unknown weaknesses in either sub-argument to be accommodated. The two sub-arguments were: (1) compliance with DO-178B; and (2) comprehensive use of formal methods.

The resulting software fitness argument was deemed compelling, and the aircraft software was authorized for use in controlled airspace.

## *Development of a COTS Component*

### Scenario

An avionics manufacturer has created a reusable module comprising a command uplink, an autopilot, and a command processing unit to be used in ultralight, low cost UASs. The module consists of a microprocessor, the associated software, a command uplink radio receiver, and a telemetry downlink transmitter. External connectors are provided so that the module can be used to control payload devices. A user-supplied configuration file customizes the module for use in each UAS.

### Argument Contract

The contract defines a set of services that the module will provide:

- Flight control
- Autopilot altitude and heading hold
- GPS-based waypoint navigation
- Sensor telemetry

The contract also guarantees the following properties, which were established by researching potential customers' needs:

- Reliability:  $\leq 10^{-5}$  failures per hour
- Dimensions:  $\leq 5 \text{ cm} \times 5 \text{ cm} \times 2 \text{ cm}$
- Mass:  $\leq 100 \text{ g}$
- Power:  $\leq 1 \text{ W}$  (when not transmitting)

Finally, the contract spells out prerequisites that must be met before its guarantees hold. The principal software-related prerequisite is that:

- Configuration data must be syntactically valid and semantically correct

### Engineering Process

The avionics module manufacturer created both the module and a fitness argument showing that it meets its contract. To support this argument, the software engineering team created the software and a corresponding software fitness argument. The software technology used includes:

- Careful semi-formal specification of the software's behavior.

- Use of a documented safer subset of C.
- Extensive use of automated static code analysis tools to catch common defects.
- Inspections and walkthroughs of critical portions of the code.
- Extensive functional testing.

### Software Authorization

The component manufacturer submitted the component and its software for pre-authorization. The authorizing agent deemed the software fitness argument compelling and granted pre-approval. Users of the component can be confident that the argument fragment associated with the component will win approval when they extend it into a complete software fitness argument by providing evidence of the configuration data's correctness.

### Conclusion

The authorization of software for the wide spectrum of UASs, the wide variety of operating environments, and the wide range of missions and payloads is a major technical challenge. The challenge cannot be met easily using any of the existing techniques such as integrity levels because such techniques are insufficiently flexible.

The approach we have described is to use rigorous software fitness arguments as the basis for authorization decisions. Building on the established technology of safety arguments, this approach can accommodate all of the variations that can be expected by tailoring the fitness argument to the aircraft, the payload, the mission, and the operating environment. Further, the technique allows diagnosis of the precise details of why authorization cannot be granted where that is the case, and thereby the technique provides both the authorizing agent and the UAS' developers with a route to making the necessary changes to permit authorization.

We presented examples to illustrate how our approach addresses UAS software authorization challenges stemming from the broad range of aircraft, operating environments, and missions and the need to authorize the use of existing software.

Space considerations preclude us from enumerating and addressing every concern that an authorizing agency might have, but we have

considered how some concerns not described here might be addressed. For example, our experience with fitness arguments suggests that *argument patterns* coupling the use of a particular software engineering technique to the claims that its use supports might help to keep authorization costs down by facilitating argument construction and easing comprehension. We have considered the use of standardized contracts to facilitate the creation and use of COTS components. We leave a more thorough exploration of these issues for future work.

### Acknowledgements

This work was funded in part by NAVAIR under contract N00421-08-1-0006 and in part by NASA under contract NAS1-02117.

### References

- [1] Kelly, Tim and Rob Weaver, 2004, The Goal Structuring Notation — A Safety Argument Notation, Proc. of Dependable Systems and Networks 2004 Workshop on Assurance Cases.
- [2] RTCA, 1992, Software Considerations in Airborne Systems and Equipment Certification, DO-178B, RTCA, Inc.
- [3] Kelly, Tim, 1998, Arguing Safety — A Systematic Approach to Managing Safety Cases, Ph.D. Thesis, Department of Computer Science, University of York, York, UK.
- [4] Adelard LLP, The Adelard Safety Case Development (ASCAD) Manual, web site, <http://www.adelard.com/web/hnav/resources/ascad/index.html>.
- [5] ISO, 1998, System and software integrity levels, ISO/IEC 15026, International Organization for Standardization.
- [6] Butler, Ricky and George Finelli, 1991, The Infeasibility of Experimental Quantification of Life-Critical Software Reliability, IEEE Transactions on Software Engineering, pp. 66–76.

*28th Digital Avionics Systems Conference  
October 25-29, 2009*

